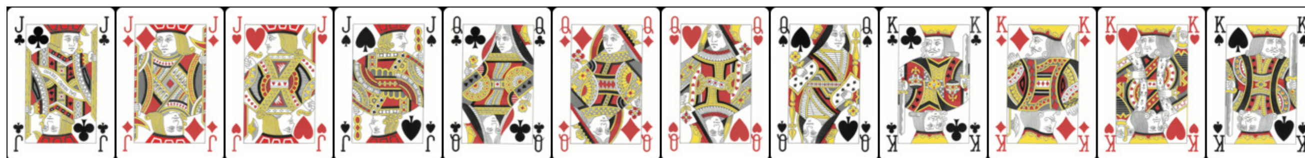
Aix-Marseille Université
Faculté des Sciences de Luminy

Licence Math-Info – Semestre 2
Lundi 27 Juin 2016

Programmation — Examen

Durée : 2 heures, Calculatrices et Documents Interdits



On souhaite générer un paquet de cartes trié, effectuer une coupe (*cut*), et un mélange américain (*riffle shuffle*).

Dans tout le devoir, pour implémenter une fonction, vous devez RÉUTILISER les fonctions des questions précédentes lorsque cela est possible, même si vous ne les avez pas implémentées.

Par ailleurs, dans tout l'examen, on utilisera pour les Booléens le type `bool`, et les constantes `true` et `false`, tous trois définis dans `<stdbool.h>`, que l'on supposera inclus comme les autres entêtes standards.

1 Cartes à jouer

On considère les deux types énumératifs `Rank` et `Suit` de l'encadré ci-dessous, définissant des énumérateurs pour les rangs et les couleurs de cartes à jouer. Les rangs (*ranks* en anglais) sont définis pour : Valet, Dame, Roi (resp. *Jack, Queen, King*), et les couleurs (*suits*) pour : trèfle, carreau, coeur, pique (resp. *club, diam, heart, spade*).

```
typedef enum Rank {
  RANK_JACK, RANK_QUEEN, RANK_KING, NB_RANKS
}  Rank;

typedef enum Suit {
  SUIT_CLUB, SUIT_DIAM, SUIT_HEART, SUIT_SPADE, NB_SUITS
} Suit;
```

Question 1. Que valent les énumérateurs : `RANK_JACK`, `NB_RANKS`, `SUIT_CLUB`, `NB_SUITS` ?

Question 2. On symbolise traditionnellement les couleurs *club, diam, heart, spade* par leurs initiales en minuscule (resp. *c, d, h, s*), et les rangs *Jack, Queen, King* par leurs initiales en majuscule (resp. *J, Q, K*). Implémenter les fonctions `Rank_SymbolChar()` et `Suit_SymbolChar()` qui renvoient respectivement le symbole d'un rang et le symbole d'une couleur dans un `char`. Utiliser un `switch` pour l'une, et consulter un tableau pour l'autre.

```
char Rank_SymbolChar (Rank rank);            /* avec switch...  */
char Suit_SymbolChar (Suit suit);            /* avec tableau... */
```

On considère le type `Card` de l'encadré ci-dessous qui définit une carte par son rang et sa couleur.

```
typedef struct Card {
  Rank rank;
  Suit suit;
} Card;
```

Question 3. Implémenter la fonction `Card_Make()` qui fabrique une carte à partir de son rang et de sa couleur.

```
Card Card_Make (Rank rank, Suit suit);
```

Question 4. On symbolise traditionnellement une carte par les symboles de son rang et de sa couleur. Par exemple, le Roi de coeur (*King of heart*) se note *Kh*. Implémenter la fonction `Card_SymbolString()` qui renvoie le symbole d'une carte sous forme d'une chaîne statique.

```
char * Card_SymbolString (Card card);
```

Question 5. Écrire un programme qui construit une carte, par exemple le Valet de pique (*Jack of spade*), la stocke dans une variable de type `Card`, puis affiche sur la sortie standard son symbole (sur l'exemple, *Js*) suivi d'un retour à la ligne. Le programme se termine sur un succès.

```
int main (void);
```

2 Paquet de cartes

On considère le type `Deck` définissant un paquet de cartes (considéré posé sur la table). Ce type est défini de telle sorte qu'on puisse ajouter/retirer facilement des cartes par le dessous et par le dessus. La position d'une carte dans un paquet est le nombre de cartes qui la précède en partant du dessous du paquet (la carte du dessous est donc par définition toujours en position 0).

```
#define DECK_CAPACITY (NB_RANKS * NB_SUITS)

typedef struct Deck {
  Card cards [DECK_CAPACITY];
  int bottom;
  int length;
} Deck;
```

Si `d` est un paquet, alors il contient `d.length` cartes. La valeur `d.length` est comprise entre 0 et `DECK_CAPACITY` inclus. Les cartes de `d` sont stockées dans le tableau `d.cards`. La carte du dessous (de position 0) se trouve à l'index `d.bottom`, c'est-à-dire dans la case `d.cards[d.bottom]`. Les cartes en positions suivantes occupent les indexes suivants. Le tableau est vu comme étant circulaire modulo `DECK_CAPACITY`, c'est-à-dire que l'index `DECK_CAPACITY-1` est conceptuellement suivi de l'index 0. Si le paquet est vide, alors `d.length` est nul et `d.bottom` est un index arbitraire qui hébergera la première carte lors d'une insertion ultérieure.

`index` :	0	1	2	3	4	5	6	7	8	9	10	11
`d.cards[index]` :	*Qh*	*Qs*								*Js*	*Qc*	*Qd*
position :	3	4	5						-1	0	1	2

(Figure A)

La figure A illustre un état possible du tableau `d.cards` pour `DECK_CAPACITY= 12` après une séquence de plusieurs insertions et suppressions. Ici, `d` contient `d.length= 5` cartes : le Valet de pique *Js* est en dessous du paquet, en position 0, correspondant à l'index `d.bottom= 9`. Il est suivi des 4 Dames *Qc*, *Qd*, *Qh* et *Qs*. La Dame de pique *Qs* occupe le dessus du paquet en position 4, correspondant à l'index 1.

`index` :	0	1	2	3	4	5	6	7	8	9	10	11
`d.cards[index]` :	*Qh*	*Qs*	**Kc**							*Js*	*Qc*	*Qd*
position :	3	4	5	6					-1	0	1	2

(Figure B)

La figure B illustre l'état du même tableau `d.cards` après l'ajout d'une 6ème carte au dessus du paquet : le Roi de trèfle *Kc*, qui prend la position `d.length= 5`, correspondant à l'index 2. Ensuite, `d.length` passe de 5 à 6.

`index` :	0	1	2	3	4	5	6	7	8	9	10	11
`d.cards[index]` :	*Qh*	*Qs*	*Kc*								*Qc*	*Qd*
position :	2	3	4	5						-1	0	1

(Figure C)

La figure C illustre l'état du même tableau `d.cards` après l'extraction de la carte du dessous, c'est-à-dire le Valet de pique (*Js*) : `d.length` passe de 6 à 5, et `d.bottom` passe de 9 à 10.

Question 6. Implémenter la fonction `Deck_InitEmpty()` qui initialise à vide un paquet passé par adresse.

```
void Deck_InitEmpty (Deck * deck);
```

Question 7. Implémenter la fonction `Deck_IsEmpty()` qui teste si un paquet est vide, ainsi que `Deck_IsFull()` qui teste si un paquet est plein (c'est-à-dire s'il a atteint la capacité de son tableau).

```
bool Deck_IsEmpty (Deck const * deck);
bool Deck_IsFull  (Deck const * deck);
```

Question 8. Implémenter la fonction `Deck_PositionIndex()` qui renvoie l'index correspondant à une position donnée dans un paquet. On souhaite que la fonction se comporte bien pour des positions allant de -1 à la longueur du paquet, c'est-à-dire allant de la position précédant immédiatement le dessous du paquet à la position suivant immédiatement le dessus du paquet. Cette condition sur la position est préalablement vérifiée avec `assert()`.

```
int Deck_PositionIndex (Deck const * deck, int position);
```

Question 9. Implémenter la fonction `Deck_AddTopCard()` qui ajoute une carte au dessus d'un paquet. La fonction vérifie préalablement avec `assert()` que le paquet n'était pas plein.

```
void Deck_AddTopCard (Deck * deck, Card card);
```

Question 10. Implémenter la fonction `Deck_RemoveBottomCard()` qui retire et renvoie la carte du dessous d'un paquet. La fonction vérifie préalablement avec `assert()` que le paquet n'était pas vide.

```
Card Deck_RemoveBottomCard (Deck * deck);
```

3 Paquet trié, coupe, et mélange américain

Question 11. Implémenter la fonction `Deck_InitSorted()` qui initialise un paquet de cartes à vide, puis le remplit avec toutes les cartes du jeu en ajoutant en boucle les cartes sur le dessus du paquet. On souhaite obtenir le paquet trié par rang (c'est-à-dire d'abord tous les valets, puis toutes les Dames, enfin tous les Rois).

```
void Deck_InitSorted (Deck * deck);
```

Question 12. Implémenter la fonction `Deck_Print()` qui affiche un paquet sur une sortie spécifiée. Les cartes sont affichées sur une ligne entre une paire de crochets suivie d'un retour à la ligne. La carte du dessous est la plus à gauche, la carte du dessus est la plus à à droite. *Cf. exemple ci-dessous.*

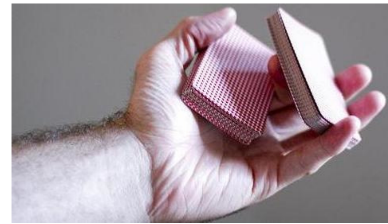
```
void Deck_Print (Deck const * deck, FILE * file);
```

```
int main (void)
{
  Deck deck;
  Deck_InitSorted (& deck);
  Deck_Print (& deck, stdout);
  return 0;
}
```

```
[ Jc Jd Jh Js Qc Qd Qh Qs Kc Kd Kh Ks ]
```

Question 13. Implémenter la fonction `Deck_MoveBottomCardsOnTop()` qui transfère `nb_cards` cartes du dessous d'un paquet `source` vers le dessus d'un paquet destination `dest`. L'ordre des cartes transférées est préservé.

```
void Deck_MoveBottomCardsOnTop (Deck * source, int nb_cards, Deck * dest);
```



Question 14. Implémenter la fonction `Deck_Cut()` qui coupe un paquet `source` en deux parties `dest1` et `dest2`. On commence par initialiser les deux paquets de destination à vide, puis les `nb_cards` cartes du dessous sont transférées dans `dest1`. Le reste est transféré dans `dest2`. (Après l'opération, le paquet `source` est donc vide).

```
void Deck_Cut (Deck * source, int nb_cards, Deck * dest1, Deck * dest2);
```

Voici un programme illustrant une coupe :

```
int main (void)
{
  Deck deck, split1, split2;
  Deck_InitSorted (&deck);
  fprintf (stdout, "deck before cut: "); Deck_Print (&deck  , stdout);

  Deck_Cut (&deck, 8, &split1, &split2);
  fprintf (stdout, "deck after  cut: "); Deck_Print (&deck  , stdout);
  fprintf (stdout, "split1: "); Deck_Print (&split1, stdout);
  fprintf (stdout, "split2: "); Deck_Print (&split2, stdout);
  return 0;
}
```

```
deck before cut: [ Jc Jd Jh Js Qc Qd Qh Qs Kc Kd Kh Ks ]
deck after  cut: [ ]
split1: [ Jc Jd Jh Js Qc Qd Qh Qs ]
split2: [ Kc Kd Kh Ks ]
```

Question 15. Implémenter la fonction `Deck_Riffle()` qui fusionne deux paquets `source1` et `source2` en un seul paquet destination `dest`. On commence par initialiser les deux paquet de destination à vide, puis les cartes des deux paquets sources sont aléatoirement entrelacées en les ventilant par le dessous comme le ferait un croupier de casino en utilisant ses deux pouces. (Après l'opération, `source1` et `source2` sont donc vides.)

```
void Deck_Riffle (Deck * dest, Deck * source1, Deck * source2);
```

Voici les étapes d'un mélange effectué par `Deck_Riffle()` où les deux sources sont issues de la coupe illustrée à la question précédente. À chaque étape, une carte est transférée du dessous d'une des deux sources (choisie aléatoirement) vers le dessus de la destination :

```
dest                    source1                     source2
                        -- -- -- -- -- -- -- --     == == == ==
[ ]                   [ Jc Jd Jh Js Qc Qd Qh Qs ] [ Kc Kd Kh Ks ]
[ Jc ]                   [ Jd Jh Js Qc Qd Qh Qs ] [ Kc Kd Kh Ks ]
[ Jc Jd ]                   [ Jh Js Qc Qd Qh Qs ] [ Kc Kd Kh Ks ]
[ Jc Jd Kc ]                [ Jh Js Qc Qd Qh Qs ]    [ Kd Kh Ks ]
[ Jc Jd Kc Jh ]                [ Js Qc Qd Qh Qs ]    [ Kd Kh Ks ]
[ Jc Jd Kc Jh Js ]                [ Qc Qd Qh Qs ]    [ Kd Kh Ks ]
[ Jc Jd Kc Jh Js Kd ]             [ Qc Qd Qh Qs ]       [ Kh Ks ]
[ Jc Jd Kc Jh Js Kd Qc ]             [ Qd Qh Qs ]       [ Kh Ks ]
[ Jc Jd Kc Jh Js Kd Qc Kh ]          [ Qd Qh Qs ]          [ Ks ]
[ Jc Jd Kc Jh Js Kd Qc Kh Qd ]          [ Qh Qs ]          [ Ks ]
[ Jc Jd Kc Jh Js Kd Qc Kh Qd Ks ]       [ Qh Qs ]             [ ]
[ Jc Jd Kc Jh Js Kd Qc Kh Qd Ks Qh ]       [ Qs ]             [ ]
[ Jc Jd Kc Jh Js Kd Qc Kh Qd Ks Qh Qs ]       [ ]             [ ]
  -- -- == -- -- == -- == -- == -- --
```